

# UNIT-2 JSP

---

ADVANCE JAVA PROGRAMMING

# TOPICS TO BE DISCUSSED

---

- Exception Handling
- Action Elements
- MVC in JSP
- JSTL
- Custom Tags.

# EXCEPTION HANDLING IN JSP

---

- The exception is normally an object that is thrown at runtime.
- Exception Handling is the process to handle the runtime errors.
- There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer.
- In JSP, there are two ways to perform exception handling:
  1. By **errorPage** and **isErrorPage** attributes of page directive
  2. By **<error-page>** element in web.xml file

# BY **ERRORPAGE** AND **ISERRORPAGE** ATTRIBUTES OF PAGE DIRECTIVE

---

- In this case, you must define and create a page to handle the exceptions, as in the error.jsp page.
- The pages where exception may occur, define the `errorPage` attribute of page directive, as in the process.jsp page.
- `index.jsp` for input values
- `process.jsp` for dividing the two numbers and displaying the result
- `error.jsp` for handling the exception

# EXCEPTION HANDLING

---

index.jsp

```
<form action="process.jsp">  
No1:<input type="text" name="n1" /><br/><br/>  
No1:<input type="text" name="n2" /><br/><br/>  
<input type="submit" value="divide"/>  
</form>
```

# EXCEPTION HANDLING

process.jsp

```
<%@ page errorPage="error.jsp" %>  
<%  
  
String num1=request.getParameter("n1");  
String num2=request.getParameter("n2");  
  
int a=Integer.parseInt(num1);  
int b=Integer.parseInt(num2);  
int c=a/b;  
out.print("division of numbers is: "+c);  
  
%>
```

# EXCEPTION HANDLING

---

error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occurred!</h3>

Exception is: <%= exception %>
```

## BY **<ERROR-PAGE>** ELEMENT IN WEB.XML FILE

---

- This approach is better because you don't need to specify the `errorPage` attribute in each jsp page.
- Specifying the single entry in the `web.xml` file will handle the exception.
- In this case, either specify `exception-type` or `error-code` with the `location` element.
- If you want to handle all the exception, you will have to specify the `java.lang.Exception` in the `exception-type` element.



# EXAMPLE

---

- There are 4 files:
  1. web.xml file for specifying the error-page element
  2. index.jsp for input values
  3. process.jsp for dividing the two numbers and displaying the result
  4. error.jsp for displaying the exception

# WEB.XML

---

```
<web-app>
```

```
  <error-page>
```

```
    <exception-type>java.lang.Exception</exception-type>
```

```
    <location>/error.jsp</location>
```

```
  </error-page>
```

```
</web-app>
```

- **This approach is better if you want to handle any exception.**

# WEB.XML

---

```
<web-app>
```

```
  <error-page>
```

```
    <error-code>500</error-code>
```

```
    <location>/error.jsp</location>
```

```
  </error-page>
```

```
</web-app>
```

- **If you know any specific error code and you want to handle that exception, specify the error-code element instead of exception-type**

# INDEX.HTML

---

```
<form action="process.jsp">
```

```
No1:<input type="text" name="n1" /><br/><br/>
```

```
No1:<input type="text" name="n2" /><br/><br/>
```

```
<input type="submit" value="divide"/>
```

```
</form>
```

# PROCESS.JSP

---

- Now there is no need to specify the `errorPage` attribute of `page` directive in the `jsp` page.

```
<%@ page errorPage="error.jsp" %>
```

```
<%
```

```
    String num1=request.getParameter("n1");
```

```
String num2=request.getParameter("n2");
```

```
    int a=Integer.parseInt(num1);
```

```
    int b=Integer.parseInt(num2);
```

```
    int c=a/b;
```

```
    out.print("division of numbers is: "+c);
```

```
    %>
```

# ERROR.JSP

---

```
<%@ page isErrorPage="true" %>
```

```
<h3>Sorry an exception occurred!</h3>
```

```
Exception is: <%= exception %>
```

# MVC-INTRODUCTION

---

- The **Model-View-Controller (MVC)** framework is an architectural/design pattern that separates an application into three main logical components **Model**, **View**, and **Controller**.
- Each architectural component is built to handle specific development aspects of an application.
- It isolates the business logic and presentation layer from each other. It was traditionally used for desktop **graphical user interfaces (GUIs)**.
- Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects.

# MVC-INTRODUCTION

---

- The main goal of this design pattern was to solve the problem of users controlling a large and complex data set by splitting a large application into specific sections that all have their own purpose.
- It is also used for designing mobile apps.
- There are two types of programming models (design models)
  1. Model 1 Architecture
  2. Model 2 (MVC) Architecture



# MVC-INTRODUCTION

---

- **Model 1 Architecture**
- Servlet and JSP are the main technologies to develop the web applications.
- **Servlet** was considered superior to CGI. Servlet technology doesn't create process, rather it creates thread to handle request. The advantage of creating thread over process is that it doesn't allocate separate memory area. Thus many subsequent requests can be easily handled by servlet.
- **Problem in Servlet technology** Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

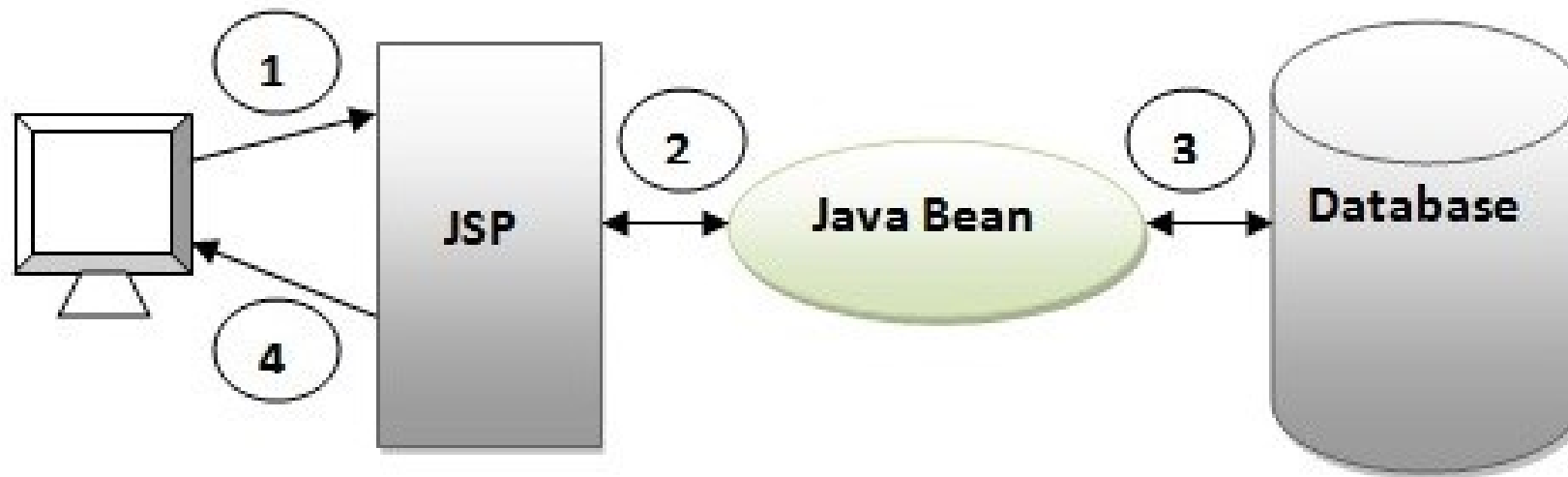
# MODEL 1 ARCHITECTURE

---

- **JSP** overcomes almost all the problems of Servlet. It provides better separation of concern, now presentation and business logic can be easily separated.
- We don't need to redeploy the application if JSP page is modified. JSP provides support to develop web application using JavaBean, custom tags and JSTL so that we can put the business logic separate from our JSP that will be easier to test and debug.

# MODEL 1 ARCHITECTURE

---



# MODEL 1 ARCHITECTURE

---

1. Browser sends request for the JSP page
2. JSP accesses Java Bean and invokes business logic
3. Java Bean connects to the database and get/save data
4. Response is sent to the browser which is generated by JSP

# MODEL 1 ARCHITECTURE

---

- **Advantage of Model 1 Architecture**
- Easy and Quick to develop web application
- **Disadvantage of Model 1 Architecture**
- **Navigation control is decentralized** since every page contains the logic to determine the next page. If JSP page name is changed that is referred by other pages, we need to change it in all the pages that leads to the maintenance problem.
- **Time consuming** You need to spend more time to develop custom tags in JSP. So that we don't need to use scriptlet tag.
- **Hard to extend** It is better for small applications but not for large applications.

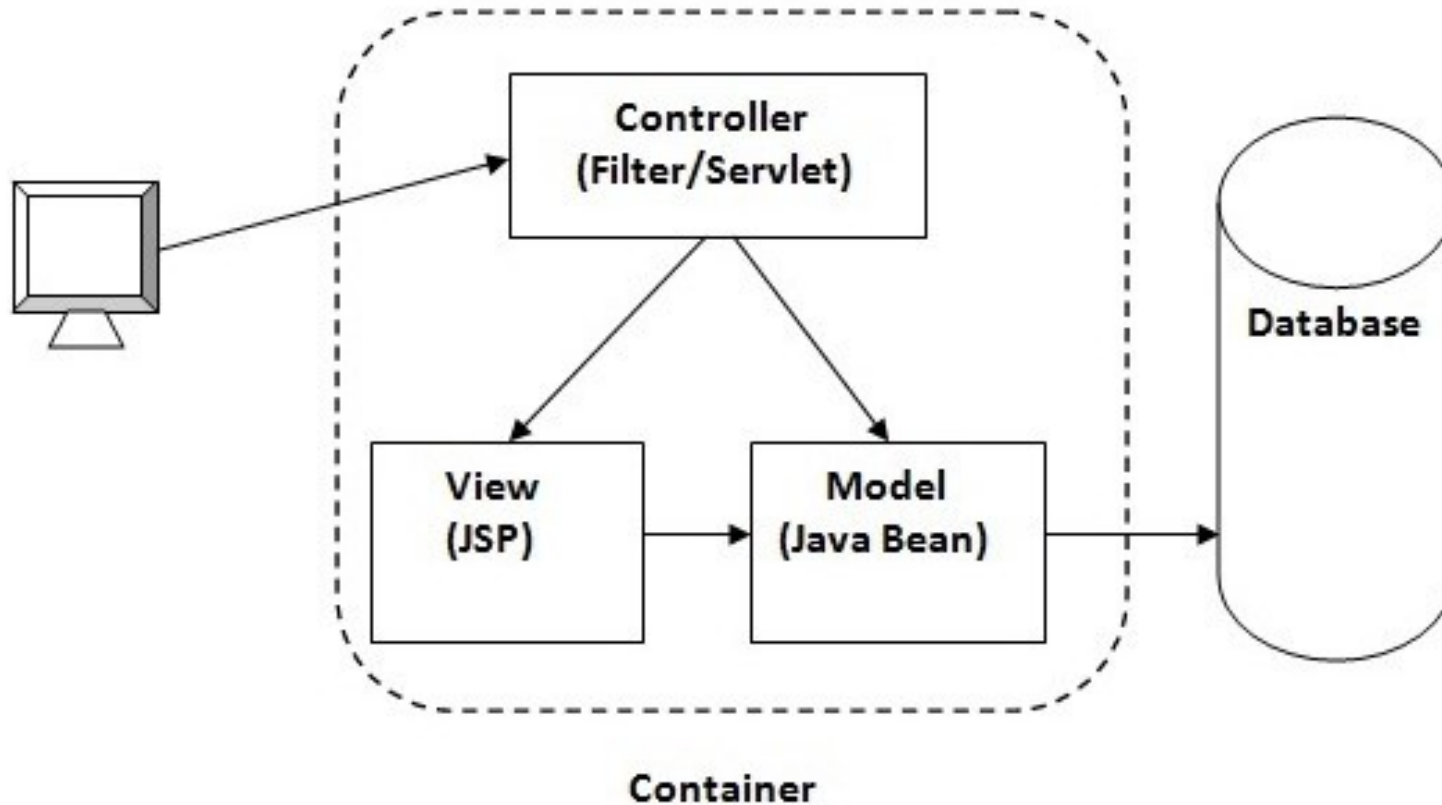
# MODEL 2 ARCHITECTURE

---

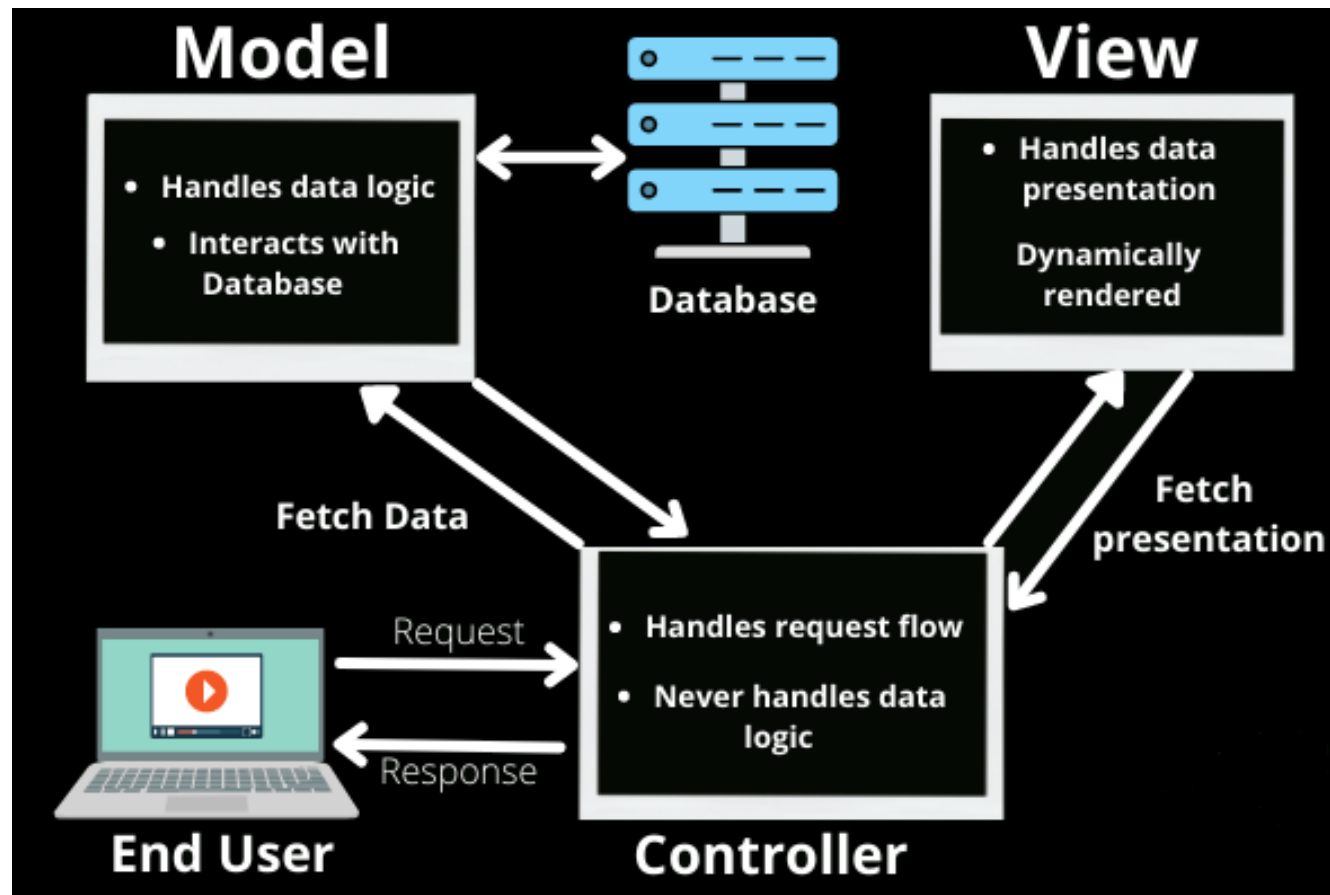
- Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.
- **Model** The model represents the state (data) and business logic of the application.
- **View** The view module is responsible to display data i.e. it represents the presentation.
- **Controller** The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

# MODEL 2 ARCHITECTURE

---



# MODEL 2 ARCHITECTURE





# MODEL 2 ARCHITECTURE

---

- **Advantage of Model 2 (MVC) Architecture**
- Navigation control is centralized Now only controller contains the logic to determine the next page.
- Easy to maintain
- Easy to extend
- Easy to test
- Better separation of concerns
- **Disadvantage of Model 2 (MVC) Architecture**
- We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

# EXAMPLE

---

- Index.jsp

```
<form action="ControllerServlet" method="post">
```

```
Name:<input type="text" name="name"><br>
```

```
Password:<input type="password" name="password"><br>
```

```
<input type="submit" value="login">
```

```
</form>
```

protected **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

---

response.setContentType("text/html");

PrintWriter out=response.getWriter();

String name=request.getParameter("name");

String password=request.getParameter("password");

LoginBean bean=**new** LoginBean();

bean.setName(name);

bean.setPassword(password);

request.setAttribute("bean",bean);

**boolean** status=bean.validate();

```
if(status){  
    RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");  
    rd.forward(request, response);  
}  
  
else{  
    RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");  
    rd.forward(request, response);  
}  
}  
  
@Override  
protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
    throws ServletException, IOException {  
    doPost(req, resp);  
}  
}
```

# LOGINBEAN

---

```
public class LoginBean {  
    private String name,password;  
  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getPassword() {  
        return password;  
    }  
}
```

```
public void setPassword(String password) {  
    this.password = password;
```

---

```
}  
  
public boolean validate(){  
    if(password.equals("admin")){  
        return true;  
    }  
    else{  
        return false;  
    }  
}  
}
```

# LOGINSUCCESS.JSP

---

```
<%@page import="com.javatpoint.LoginBean"%>
```

```
<p>You are successfully logged in!</p>
```

```
<%
```

```
LoginBean bean=(LoginBean)request.getAttribute("bean");
```

```
out.print("Welcome, "+bean.getName());
```

```
%>
```

# LOGINERROR.JSP

---

```
<p>Sorry! username or password error</p>
```

```
<%@ include file="index.jsp" %>
```



# JDBC CONNECTIVITY WITH JSP

---

- A JDBC (Java Database Connectivity) is being utilized to connect Java Server Pages with a database in a JSP database connection.
- JDBC acts as an essential connector that allows your JSP pages to communicate with a database.
- A Java API is utilized to handle interactions with databases through Java methods as well as SQL queries.
- This feature is essential for developing interactive web applications that need data storage, retrieval, and manipulation.

# JDBC CONNECTIVITY WITH JSP

---

```
<%@ page import="java.sql.*" %>
```

```
<html> <head><title>Database Connection Example</title></head>
```

```
<body>
```

```
<% Connection con = null; Statement stmt = null; ResultSet rs = null;
```

```
try { // Load JDBC Driver
```

```
Class.forName("com.mysql.jdbc.Driver"); // Establish Connection
```

```
con = DriverManager.getConnection(
```

```
"jdbc:mysql://localhost:3306/DatabaseName", "username", "password");
```

# JDBC CONNECTIVITY WITH JSP

---

```
stmt = con.createStatement();  
// Execute Query  
rs = stmt.executeQuery("SELECT * FROM users");  
// Process Result Set  
while(rs.next()) {  
    out.println("Name: " + rs.getString("name") + "<br>");  
}  
  
catch (ClassNotFoundException e) {  
    out.println("Driver not found: " + e.getMessage());  
}  
catch (SQLException e) { out.println("SQL Error: " + e.getMessage()); }
```

# JDBC CONNECTIVITY WITH JSP

---

- finally { // Close resources
- try { if (rs != null) rs.close();
- if (stmt != null) stmt.close();
- if (con != null) con.close();
- } catch (SQLException e) {
- out.println("Closing Error: " + e.getMessage());
- } } %>
- </body> </html>

# JSTL

---

- The JSTL, which stands for JavaServer Pages Standard Tag Library, comprises practical JSP tags that encompass fundamental features found in various JSP applications.
- JSTL offers assistance for basic tasks like loops and if statements, tags for handling XML files, internationalization tags, and SQL tags.
- It offers a structure for merging the current custom tags with the JSTL tags.

# INSTALL JSTL LIBRARY

---

- To begin working with JSP tags you need to first install the JSTL library. If you are using the Apache Tomcat container, then follow these two steps –
- **Step 1** – Download the binary distribution from [Apache Standard Taglib](#) and unpack the compressed file.
- **Step 2** – To use the Standard Taglib from its **Jakarta Taglibs distribution**, simply copy the JAR files in the distribution's 'lib' directory to your application's **webapps\ROOT\WEB-INF\lib** directory.
- To use any of the libraries, you must include a `<taglib>` directive at the top of each JSP that uses the library.

# CLASSIFICATION OF THE JSTL TAGS

---

- The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page –
- **Core Tags**
- **Formatting tags**
- **SQL tags**
- **XML tags**
- **JSTL Functions**

# CORE TAGS

---

- The core group of tags are the most commonly used JSTL tags.
- Following is the syntax to include the JSTL Core library in your JSP –
- **<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>**



# CORE TAGS

S.No.	Tag & Description
1	<b>&lt;c:out&gt;</b> Like <code>&lt;%= ... &gt;</code> , but for expressions.
2	<b>&lt;c:set &gt;</b> Sets the result of an expression evaluation in a ' <b>scope</b> '
3	<b>&lt;c:remove &gt;</b> Removes a <b>scoped variable</b> (from a particular scope, if specified).
4	<b>&lt;c:catch&gt;</b> Catches any <b>Throwable</b> that occurs in its body and optionally exposes it.
5	<b>&lt;c:if&gt;</b> Simple conditional tag which evalutes its body if the supplied condition is true.

# FORMATTING TAGS

---

- The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Websites.
- Following is the syntax to include Formatting library in your JSP –
- **<%@ taglib prefix = "fmt" uri =  
"http://java.sun.com/jsp/jstl/fmt" %>**

# FORMATTING TAGS

S.No.	Tag & Description
1	<b>&lt;fmt:formatNumber&gt;</b> To render numerical value with specific precision or format.
2	<b>&lt;fmt:parseNumber&gt;</b> Parses the string representation of a number, currency, or percentage.
3	<b>&lt;fmt:formatDate&gt;</b> Formats a date and/or time using the supplied styles and pattern.
4	<b>&lt;fmt:parseDate&gt;</b> Parses the string representation of a date and/or time
5	<b>&lt;fmt:bundle&gt;</b> Loads a resource bundle to be used by its tag body.
6	<b>&lt;fmt:setLocale&gt;</b> Stores the given locale in the locale configuration variable.

# SQL TAGS

---

- The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as **Oracle**, **mySQL**, or **Microsoft SQL Server**.
- Following is the syntax to include JSTL SQL library in your JSP –
- **<%@ taglib prefix = "sql" uri =  
"http://java.sun.com/jsp/jstl/sql" %>**

# SQL TAGS

S.No.	Tag & Description
1	<b>&lt;sql:setDataSource&gt;</b> Creates a simple DataSource suitable only for prototyping
2	<b>&lt;sql:query&gt;</b> Executes the SQL query defined in its body or through the sql attribute.
3	<b>&lt;sql:update&gt;</b> Executes the SQL update defined in its body or through the sql attribute.
4	<b>&lt;sql:param&gt;</b> Sets a parameter in an SQL statement to the specified value.
5	<b>&lt;sql:dateParam&gt;</b> Sets a parameter in an SQL statement to the specified java.util.Date value.
6	<b>&lt;sql:transaction &gt;</b> Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

# **XML TAGS**

---

- The JSTL XML tags provide a JSP-centric way of creating and manipulating the XML documents. Following is the syntax to include the JSTL XML library in your JSP.
- The JSTL XML tag library has custom tags for interacting with the XML data. This includes parsing the XML, transforming the XML data, and the flow control based on the XPath expressions.
- **<%@ taglib prefix = "x" uri =  
"http://java.sun.com/jsp/jstl/xml" %>**

# XML TAGS

1	<b>&lt;x:out&gt;</b> Like <code>&lt;%= ... &gt;</code> , but for XPath expressions.
2	<b>&lt;x:parse&gt;</b> Used to parse the XML data specified either via an attribute or in the tag body.
3	<b>&lt;x:set &gt;</b> Sets a variable to the value of an XPath expression.
4	<b>&lt;x:if &gt;</b> Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.
5	<b>&lt;x:forEach&gt;</b> To loop over nodes in an XML document.

# JSTL FUNCTIONS

---

- JSTL includes a number of standard functions, most of which are common string manipulation functions.
- Following is the syntax to include JSTL Functions library in your JSP –
- **<%@ taglib prefix = "fn" uri =  
"http://java.sun.com/jsp/jstl/functions" %>**



# JSTL FUNCTIONS

---

1	<b>fn:contains()</b> Tests if an input string contains the specified substring.
2	<b>fn:containsIgnoreCase()</b> Tests if an input string contains the specified substring in a case insensitive way.
3	<b>fn:endsWith()</b> Tests if an input string ends with the specified suffix.
4	<b>fn:escapeXml()</b> Escapes characters that can be interpreted as XML markup.
5	<b>fn:indexOf()</b> Returns the index withing a string of the first occurrence of a specified substring.

# EXAMPLE OF CORE TAGS

---

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<html>
```

```
<head>
```

```
<title>Tag Example</title>
```

```
</head>
```

```
<body>
```

```
<c:out value="${'Welcome to Uttaranchal University}"/>
```

```
</body>
```

```
</html>
```

# EXAMPLE OF FORMATTING TAGS

---

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<html>

<head>

<title>fmt:parseNumber tag</title>

</head>

<body>

<h3>The fmt:parseNumber tag Example is:</h3>

# EXAMPLE OF FORMATTING TAGS

---

```
<c:set var="Amount" value="786.970" />
```

```
<fmt:parseNumber var="j" type="number" value="${Amount}" />
```

```
<p><i>Amount is:</i> <c:out value="${j}" /></p>
```

```
<fmt:parseNumber var="j" integerOnly="true" type="number" value="${Amount}" />
```

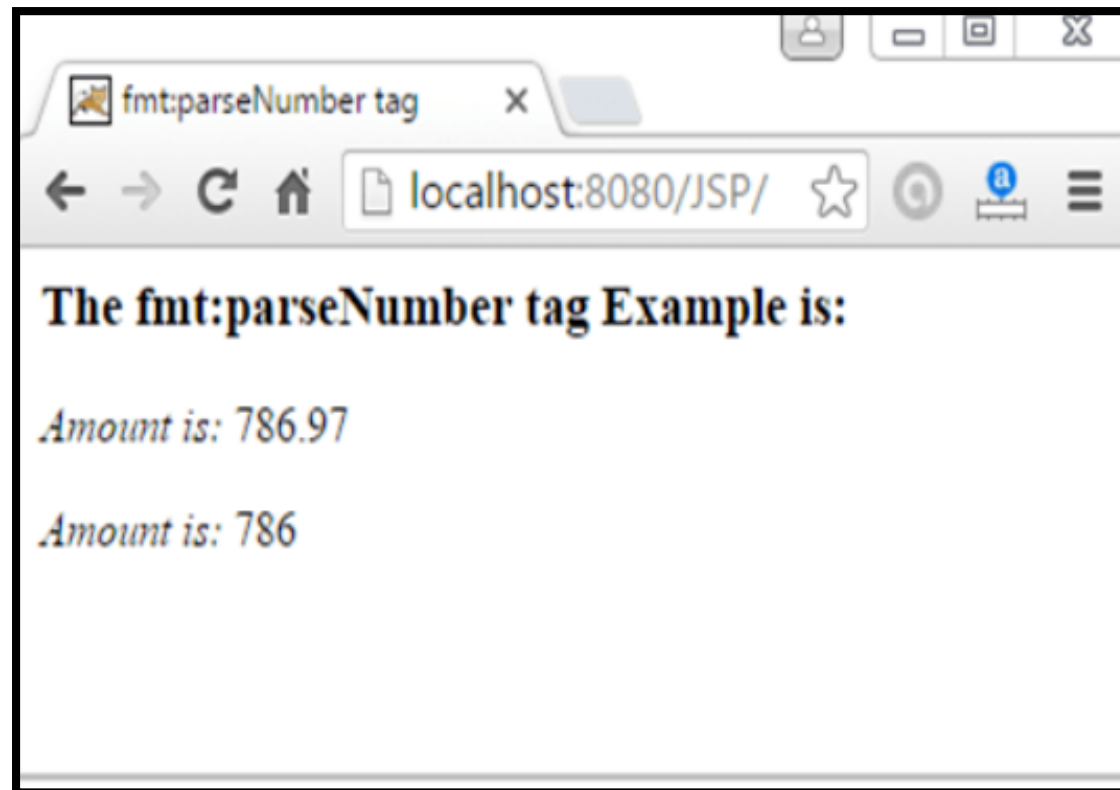
```
<p><i>Amount is:</i> <c:out value="${j}" /></p>
```

```
</body>
```

```
</html>
```

# EXAMPLE OF FORMATTING TAGS-OUTPUT

---



# EXAMPLE OF SQL TAGS

---

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
```

```
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
```

```
<html>
```

```
<head>
```

```
<title>sql:query Tag</title>
```

```
</head>
```

```
<body>
```

```
    <sql:setDataSource var="db" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/test" user="root" password="1234"/>
```

# EXAMPLE OF SQL TAGS

---

```
<sql:query dataSource="${db}" var="rs">
```

```
SELECT * from Students;
```

```
</sql:query>
```

```
  <table border="2" width="100%">
```

```
  <tr> <th>Student ID</th>
```

```
  <th>First Name</th>
```

```
  <th>Last Name</th>
```

```
  <th>Age</th>
```

```
</tr>
```

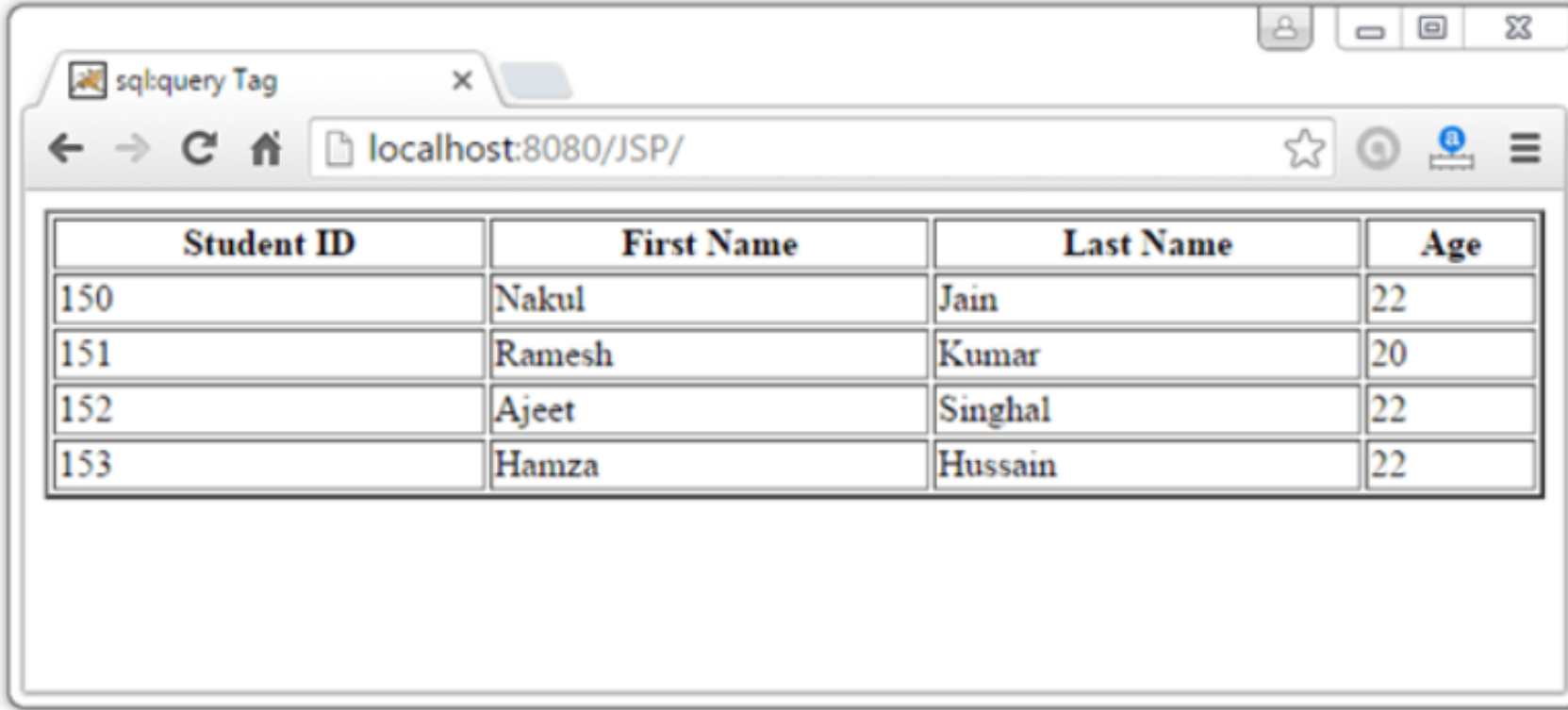
# EXAMPLE OF SQL TAGS

---

```
<c:forEach var="table" items="${rs.rows}">
<tr>
<td><c:out value="${table.id}"/></td>
<td><c:out value="${table.First_Name}"/></td>
<td><c:out value="${table.Last_Name}"/></td>
<td><c:out value="${table.Age}"/></td>
</tr> </c:forEach>
</table>
</body>
</html>
```



# EXAMPLE OF SQL TAGS- OUTPUT



Student ID	First Name	Last Name	Age
150	Nakul	Jain	22
151	Ramesh	Kumar	20
152	Ajeet	Singhal	22
153	Hamza	Hussain	22

# EXAMPLE OF XML TAGS

---

```
<%@ taglib prefix="c" uri="http://java.sun.com
/jsp/jstl/core" %>
```

```
<%@ taglib prefix="x" uri="http://java.sun.co
m/jsp/jstl/xml" %>
```

```
<html>
<head>
<title>XML Tags</title>
</head>
<body>
<h2>Vegetable Information:</h2>
<c:set var="vegetable">
<vegetables>
```

```
<vegetable>
  <name>onion</name>
  <price>40/kg</price>
</vegetable>
<vegetable>
  <name>Potato</name>
  <price>30/kg</price>
</vegetable>
<vegetable>
  <name>Tomato</name>
  <price>90/kg</price>
</vegetable>
</vegetables>
```

# EXAMPLE OF XML TAGS

---

**</c:set>**

**<x:parse xml="{vegetable}" var="output"/>**

**<b>Name of the vegetable is</b>:**

**<x:out select="\$output/vegetables/vegetable[1]/name" /><br>**

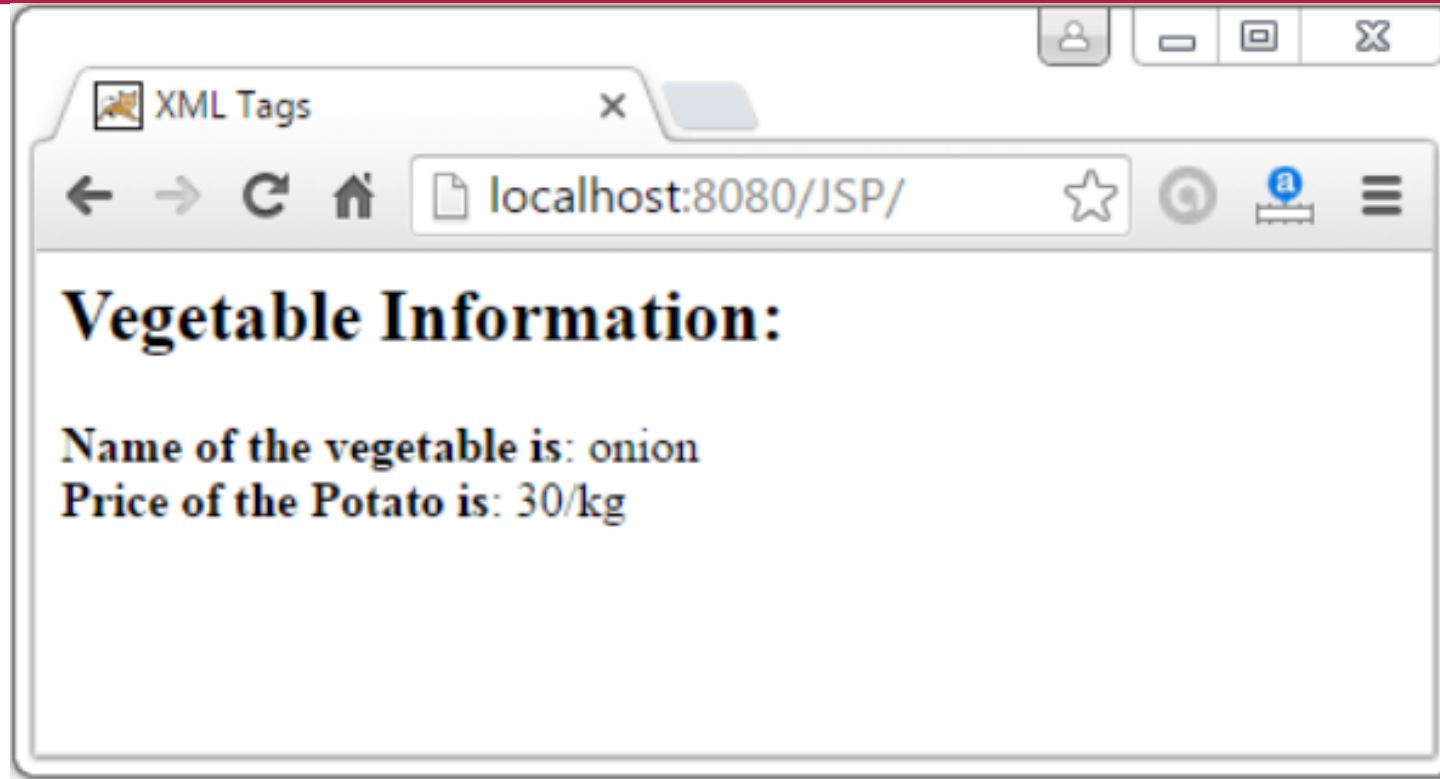
**<b>Price of the Potato is</b>:**

**<x:out select="\$output/vegetables/vegetable[2]/price" />**

**</body>**

**</html>**

# EXAMPLE OF XML TAGS-OUTPUT



# EXAMPLE OF JSTL FUNCTIONS

---

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

```
<html>
```

```
<body>
```

```
<c:set var="Book" value="Programming with Java"/>
```

```
<c:set var="author" value="E Balaguruswamy"/>
```

Hi, This is **`${fn:toUpperCase(Book)}`** written by **`${fn:toUpperCase(author)}`**.

```
</body>
```

```
</html>
```

# EXAMPLE OF JSTL FUNCTIONS- OUTPUT

---

- **Hi, This is PROGRAMMING IN JAVA written by E BALAGURUSWAMY.**

# CUSTOM TAGS

---

- Custom tags in Java Server Pages are action tags created by users.
- Each tag is linked with a specific tag handler to carry out the necessary functions.
- Hence, it divides the business logic from JSP and prevents the necessity of utilizing scriptlet tags.
- The use of scriptlet tag in JSP pages makes it hard to comprehend due to embedded Java code and should be avoided.

# SCRIPTLET VERSUS CUSTOM TAGS

---

1. Custom tags help in segregating the business logic from Java Server Pages. JSP still manages the flow while the processing is passed on to a distinct Java class (tag handler class).
2. It enhances the readability of Java Server Pages. JSPs without scripts are much simpler to comprehend than JSPs containing scriptlets.
3. Using tags instead of writing Java code within JSPs is a simpler option for frontend developers.
4. Custom tags can be utilized in different applications, but it is impossible to recycle the Java code included in a JSP using scriptlets.



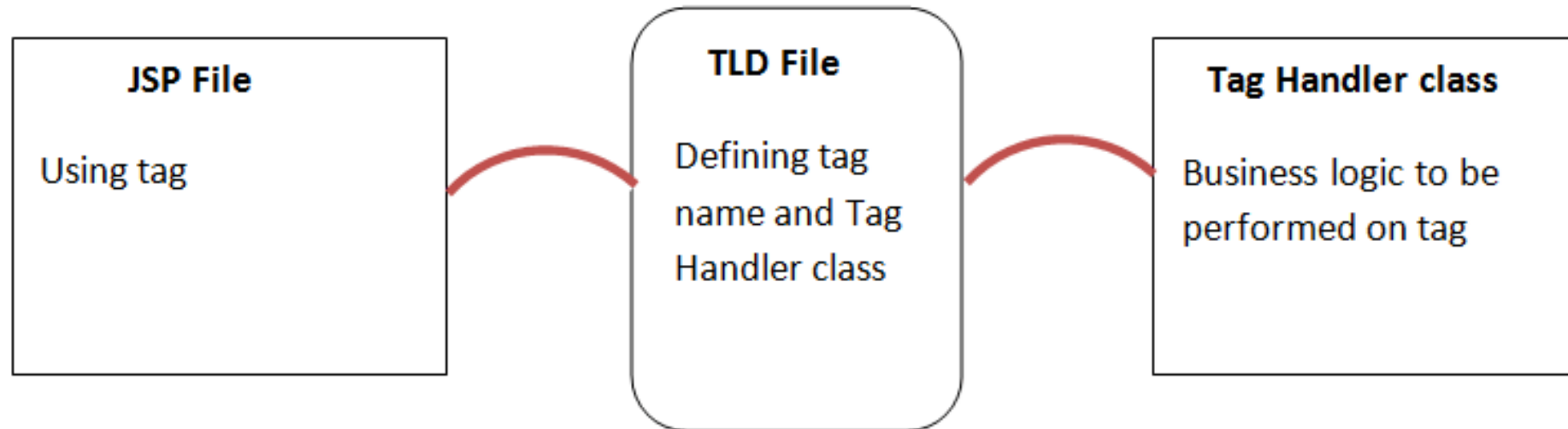
# CREATING CUSTOM TAGS

---

- For creating any custom tag, we need to follow following steps:
  - 1. Create the Tag handler class** and perform action at the start or at the end of the tag.
  - 2. Create the Tag Library Descriptor (TLD) file** and define tags
  - 3. Create the JSP file** that uses the Custom tag defined in the TLD file

# CREATING CUSTOM TAGS

---



# CREATING CUSTOM TAGS

---

## 1) Create the Tag handler class

- To create the Tag Handler, we are inheriting the **TagSupport** class and overriding its method **doStartTag()**.
- To write data for the jsp, we need to use the **JspWriter** class.
- The **PageContext** class provides **getOut()** method that returns the instance of JspWriter class.
- TagSupport class provides instance of pageContext by default.

# CREATING CUSTOM TAGS

---

## 2) Create the TLD file

- **Tag Library Descriptor (TLD)** file contains information of tag and Tag Handler classes. It must be contained inside the **WEB-INF** directory.

## 3) Create the JSP file

- It uses **taglib** directive to use the tags defined in the tld file.

# EXAMPLE OF CUSTOM TAGS-MYTAGHANDLER.JAVA

---

```
import java.util.Calendar;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
public class MyTagHandler extends TagSupport{
    public int doStartTag() throws JspException {
        JspWriter out=pageContext.getOut();//returns the instance of JspWriter
        try{    out.print(Calendar.getInstance().getTime());//printing date and time using JspWriter
        }catch(Exception e){System.out.println(e);}
        return SKIP_BODY;//will not evaluate the body content of the tag
    } }
```

# EXAMPLE OF CUSTOM TAGS- MYTAGS.TLD

---

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<!DOCTYPE taglib
```

```
    PUBLIC "-
```

```
//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN" "http://java.sun.com/j2ee  
/dtd/web-jsptaglibrary_1_2.dtd">
```

```
<taglib>
```

```
    <tlib-version>1.0</tlib-version>
```

```
    <jsp-version>1.2</jsp-version>
```

```
    <short-name>simple</short-name>
```

# EXAMPLE OF CUSTOM TAGS- MYTAGS.TLD

---

```
<uri>http://tomcat.apache.org/example-taglib</uri>
```

```
<tag>
```

```
<name>today</name>
```

```
<tag-class>com.javatpoint.sonoo.MyTagHandler</tag-class>
```

```
</tag>
```

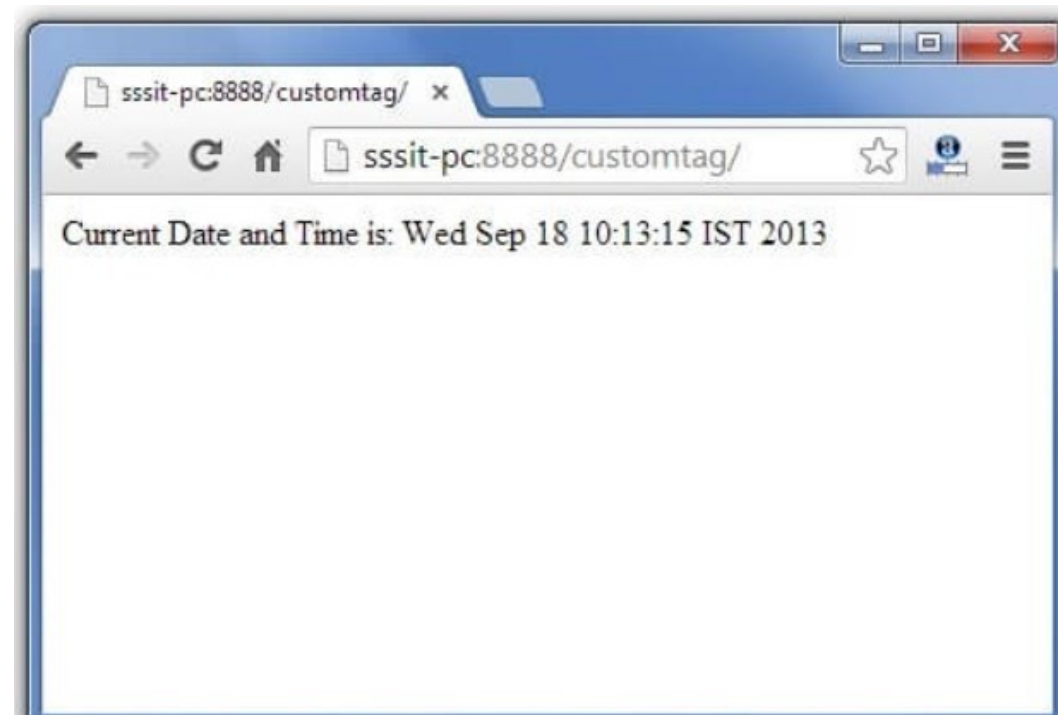
```
</taglib>
```

# EXAMPLE OF CUSTOM TAGS-INDEX.JSP

---

`<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>`

Current Date and Time is: `<m:today/>`





# ATTRIBUTES IN CUSTOM TAGS

---

- There can be defined too many attributes for any custom tag. To define the attribute, you need to perform two tasks:
  1. Define the property in the TagHandler class with the attribute name and define the setter method
  2. define the attribute element inside the tag element in the TLD file

# EXAMPLE OF CUSTOM TAG WITH ATTRIBUTE

---

- Index.jsp  
<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>

Cube of 4 is: <m:cube number="4"></m:cube>

## CubeNumber.java

```
import javax.servlet.jsp.JspException;  
import javax.servlet.jsp.JspWriter;  
import javax.servlet.jsp.tagext.TagSupport;  
  
public class CubeNumber extends TagSupport{  
    private int number;
```

# EXAMPLE OF CUSTOM TAG WITH ATTRIBUTE

---

```
public void setNumber(int number) {  
    this.number = number;  
}  
  
public int doStartTag() throws JspException {  
    JspWriter out=pageContext.getOut();  
    try{  
        out.print(number*number*number);  
    }catch(Exception e){e.printStackTrace();}  
  
    return SKIP_BODY;  
}  
}
```

# MYTAGS.TLD

---

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<!DOCTYPE taglib
```

```
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
```

```
    "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">
```

```
<taglib>
```

```
    <tlib-version>1.0</tlib-version>
```

```
    <jsp-version>1.2</jsp-version>
```

```
    <short-name>simple</short-name>
```

```
    <uri>http://tomcat.apache.org/example-taglib</uri>
```

```
    <description>A simple tag library for the examples</description>
```

# MYTAGS.TLD

---

<tag>

<name>cube</name>

<tag-class>com.javatpoint.taghandler.CubeNumber</tag-class>

<attribute>

<name>number</name>

<required>true</required>

</attribute>

</tag>

</taglib>

# OUTPUT

---

Output

Cube of 4 is: 64